Remeex: Melody Extraction by Machine Learning

Justin Mao-Jones
Center for Data Science
New York University
justinmaojones@nyu.edu

Junbo Zhao
Center for Data Science
New York University
j.zhao@nyu.edu

Rita Li
Center for Data Science
New York University
ml4713@nyu.edu

Final Report: DSGA-1003 Machine Learning

1 Introduction

In the project, we aim to build and evaluate machine learning models for the task of extracting melody from digital music files. Melody extraction is an active research area in the Music Information Retrieval research community [1]. It has many applications, such as query by humming (e.g. hum a song into your phone and an app tells you what song it is), cover song identification, genre classification, and mood classification.

Generally speaking, melody is the predominant pitch in a piece of music that captures the essence of a song. As a motivating example, the melody is the tune one might hum when asked "what does the song sound like?" Unfortunately, there does not seem to be a precise definition of melody. As a machine learning task, we require some sort of precise definition, and for this project we adopt the definition described by [1]:

> Melody is the fundamental frequency [1] from musical content with a lead voice or instrument. Melody extraction is the estimation of this melody from a single source.

While this definition is still open to interpretation, it can be used by human experts to generate the melody of a piece of music. Note that we constrain the definition of melody extraction to a single source, meaning that the melody is only coming from a single lead voice or instrument. The motivation for this simplification is that it can make the task of melody extraction easier.

Melody extraction is a supervised learning task and requires a set of labeled data. For this project, we used MedleyDB [2], a dataset of annotated, royalty-free multitrack recordings.

Many approaches to melody extraction have been attempted [1], including pure signal processing [3], dynamic programming [4], support vector machines(SVM)[5], and Hidden Markov Models (HMM) [6] [8]. In this project, our best performing model was a recurrent neural network (RNN) with long short-term memory cells (LSTM). We also experimented with SVM, HMM, and Convolutional Neural Networks.

[1] Any audio signal can be represented as a sum of a series of sinusoids. The fundamental frequency of a signal defined as the lowest frequency in the series. It can be derived through the Fourier Transform.

2 Problem Definition

Melody extraction has two primary tasks, which can be done together or separately:

1. Voicing detection (i.e. classification of whether or not the melody is present),
2. Melody pitch tracking.

The first task derives from the fact that sometimes there is no melody. The tasks can be approached in separate algorithms or in the same algorithm.

Melody pitch tracking is the task of identifying the predominant frequency during a *frame* (a small time interval). We are effectively constrained to the frame size used in the MedleyDB melody annotations, and thus our frames are roughly 4ms.

An important modeling question is whether to predict melody labels through regression or classification. Regression can seem to be a natural fit, because frequency lies on a continuous spectrum. However, human music tends to be composed on a discrete scale, i.e. musical notes. Predictions could be refined by "rounding" the predictions to the nearest note. As a classification task, we can model all possible notes, such as the 88 keys found on a standard piano. Typically, most songs may not even cover this entire spectrum, and so we can reduce the scope of our classification to only those notes present in the data.

We experimented with both regression and classification. Regression was used in SVM experiments while a classification was used in all other modeling approaches.

2.1 Melody Specific Challenges

There are two major difficulties with melody. First, the melody is often mixed in with other musical signals. For example, the melody might be produced by the singer, but there is also a guitar, a bass guitar, and drum beats playing at the same time.

Second, it is likely that melody is context dependent. For example, it might actually be impossible for any algorithm to detect discernable melody patterns in a 4 ms window without any additional information. Intuitively, it would seem that one must listen to the music surrounding that specific time frame in order to determine the melody. Thus, the features of a specific sample instance should include audio signal information from surrounding time frames. The number of such features could be potentially large, possibly huge. How large of a time window is needed to identify a melody? An additional related question is whether or not future information is required to predict melodies.

3 Data

MedleyDB [2] consists of 108 multitracks, including stereo quality mixed audio, with melody annotations and stems[2]. The multitracks include songs from a variety of genres, including Singer/Songwriter, Classical, Rock, World/Folk, Fusion, Jazz, Pop, Musical Theatre, and Rap. In total there are approximately 380 minutes of audio.

The audio files are provided in WAV format (44.1 kHz, 16 bit). In other words, each audio file contains 44,100 digital audio samples per second. Each audio file is accompanied by a single source melody annotation, provided in csv format. A melody label is a number that represents the predominant frequency over a pre-defined window of time. There are 172 melody labels per second. Each melody label overlaps roughly 256 audio samples.

One of the benefits of MedleyDB is that it was carefully curated to provide a complete set of melody annotations combined with high-quality songs. Thus, we are operating under the assumption that we do not have missing data and that the melody annotations perfectly overlap with the audio samples. Unfortunately, it would not be practically feasible to check this assumption thoroughly.

[2] In a recording session, there is a separate microphone for each instrument (or sets of instruments), and thus there are separate recordings. For example, the singer is recorded separately from the guitar. A stem is one of these separate recordings. When added together the mix will sound like a complete song.

A question we explored throughout the project is whether or not we have enough data. On the one hand, 108 songs does not sound large, on the other hand, 172 melody labels per second, at an average of 3 minutes per song, corresponds to over 3 million training samples. So the question is whether or not the data contains enough variety for the models to generalize well to new songs.

3.1 Data Pre-processing

MedleyDB comes with an accompanying API for working with its data files, and so very little data pre-processing was required. The majority of data processing work was in the form of feature generation, such as Constant-Q Transforms (CQT), Short-time Fourier Transform(STFT), Mel-frequency cepstral coefficients (MFCCs) and generating training, validation, and test splits.

4 Evaluation Metrics For Model Performance

Given that melody extraction consists of two tasks, it is natural to evaluate each task both separately and together, thus yielding three different evaluation metrics. Typically, researchers use accuracy to evaluate performance [3], and so we will do the same here by following the MIREX accuracy definitions, with one small modification:

1. Raw pitch accuracy - probability of a correct pitch value (within 1/4 tone[4]) given that the frame is voiced,
2. Overall accuracy = $\frac{TPC+TN}{TO}$

   TP = total number of frames correctly predicted as voiced

   TPC = total number of TP frames in which pitch was also correctly predicted

   TN = total number of frames correctly predicted as unvoiced

   TO = total number of predictions

Our modification has to do with voicing detection. Throughout the project, we take a general strategy of first implementing our methods to the voicing detection task, and then to overall melody extraction. The reasoning is that we presumed the former to be a simpler task than pitch tracking. For the voicing detection tasks, TPR would not be enough to accurately represent the ability of a machine learning algorithm to predict the presence of melody. Thus, we instead use voicing detection accuracy, which is the total number of correct predictions divided by the total number of predictions.

For overall melody extraction, we look at a combination of raw pitch accuracy and overall accuracy, as this seems to be the common approach in the MIR community.

5 Training/Validation/Testing

We split our training, validation, and testing sets on songs. In other words, no songs overlap between sets. This intuitively makes sense. In a real world application, a melody extraction procedure would be applied to songs it had never seen before.

Given the limited number of songs in our dataset, a possible approach would be to use cross-validation. However, running cross-validation on deep-learning architectures would be computationally expensive and not practical.

Our splits consisted of 27 songs for the test set, 27 songs for validation, and the remaining 54 songs for training. The songs were randomly chosen for each set. The sets were fixed at the beginning of the project so that different models could be evaluated on the same data.

In all there are 1,911,637 training labels, 781,458 validation labels, 743,368 testing labels.

[3] http://www.music-ir.org/mirex/wiki/2014:Audio_Melody_Extraction

[4] a tone is a step between two keys on a piano; e.g. G is one tone higher than F.

6 Feature Extraction

Since Music Information Retrieval (MIR) is closely related to the Speech Recognition (SR) community, it naturally absorbs feature descriptors and machine learning techniques from the Speech community, which has a relatively larger amount of literature. We briefly studied the marriage between the communities and settled on a few methods that could be appropriate.

- **Raw Audio**. It is a digital representation of analog signal from each time stamp and within each frame, the value is normalized to represent the amplitude of the original signal. Since it involves no extraction process, we include this feature in our baseline model and we have an 128-D vector for each time stamp.
- **STFT**. The short-time Fourier transform is a widely used signal preprocessing technique. In general, the signal is chunked into frames where STFT is applied to each chunked frame, with a window length typically assigned as some short window sizes[1]. Frames can overlap in a sliding window fashion. Particularly, we adopt 1024-point discrete fourier transform on each short-time window, by which we attempt to sample the frequency domain uniformly with 1024 points. The STFT was generated so that the time frames aligned exactly with the melody annotations, which is easily checked by verifying that the length along the time dimension of the STFT matrix matches the length of the melody label vector.
- **CQT**. The constant Q transform utilizes a series of logarithmically spaced filters to transform data series into the frequency domain with the k-th filter having a spectral width some multiple of the previous filter's width. See Figure 1 for an example illustration. The intuition behind adjusting the filter width is that higher frequencies have smaller wave lengths, and thus require smaller windows to achieve the same resolution.

  The CQT we construct on our data is represented as a numerical vector with length 84, which seemed sufficiently large enough to capture the relevant spectrum of energies of all songs. For example, almost all frequencies playable on a piano, except the very highest, would be captured in this CQT. We adjusted the hop size of the CQT so that the output has the same number of time steps as the melody labels. In addition, the CQT was constructed so that the CQT time frames aligned exactly with the melody annotations. In other words, both CQT and melody annotations contained 172 time frames per second.

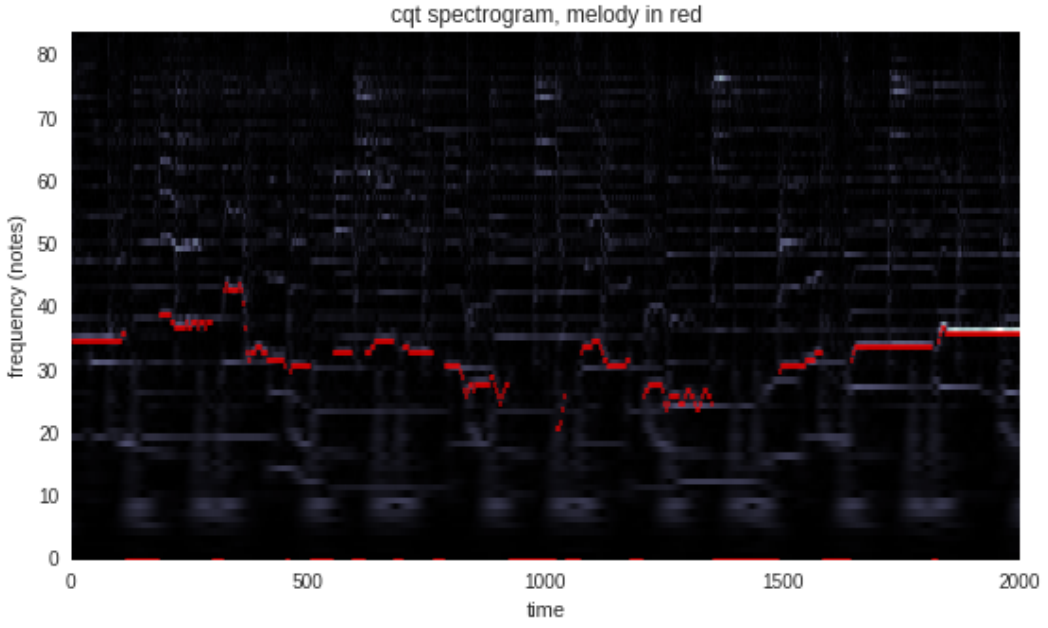


Figure 1: CQT heatmap of part of a Beatles song with the melody overlaid in red. White indicates energy, black indicates no energy.

- **MFCCs**. The Mel-frequency cepstral coefficients have been the dominant feature descriptor in Speech community over the past 30 years. It is basically a linear cosine encoding of the log power spectrum on a Mel-scale of frequency, which is biologically inspired by the inner workings of the human ear. The MFCC is extracted into a vector with length 20. As with the STFT and CQT, the time frames of the MFCC align exactly with the melody annotations.

In general, Fourier transform intuitively seems to be natural to the problem of melody extraction. The human auditory system naturally performs a Fourier analysis in the conversion of sound to neural impulses. In addition, the Fourier transform generates a time series of frequency vectors, and the melody traverses across these vectors through time.

We used the Python package `librosa`[5] for all feature generation tasks.

7 Baseline Models

In any machine learning task, it is critical to establish a baseline for comparison against model results. We have explored several baseline models for both voicing detection and pitch estimation.

- **Majority Vote**

  The crudest baseline is is to predict the majority class. The left side of Figure 2 shows a histogram of all melody labels. The majority class is class 0, i.e. 'no melody', which comprises 55% of the label distribution. Within the distribution of melody notes, as shown in the right side of Figure 2, the majority class is 38 'D4', representing around 7% of notes. This simple baseline applies to both voicing detection and overall melody extraction, with the same accuracy for each scope.

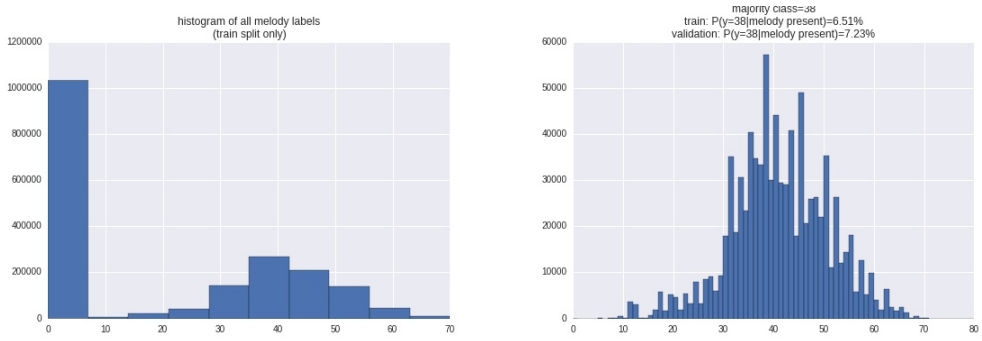


Figure 2: Melody Label Distribution

One of the primary difficulties with deriving a better baseline is that classification of melody v.s. no melody when music is playing is quite hard, even for machine learning algorithms, especially given that 'no melody' is the majority class. Thus, this crude method serves as our baseline.

- **Loudest STFT Frequency**

  Another simple baseline that we experimented with was to use the loudest frequency at each time step. We did this by taking the dominated value along the frequency domain of an STFT for each song.

  The overall accuracy of voicing detection by baseline approach is 47.25%. The overall accuracy is worse than predicting the majority class. Figure 3 shows the histograms of both true positive rates and false positive rates from each song, i.e. STFT shows the existence of melody while in fact there isnt. As we could see, the false positive rates are high, signifying that STFT tends to respond true regardless of whether there is melody existed.

[5] http://bmcfee.github.io/librosa/#

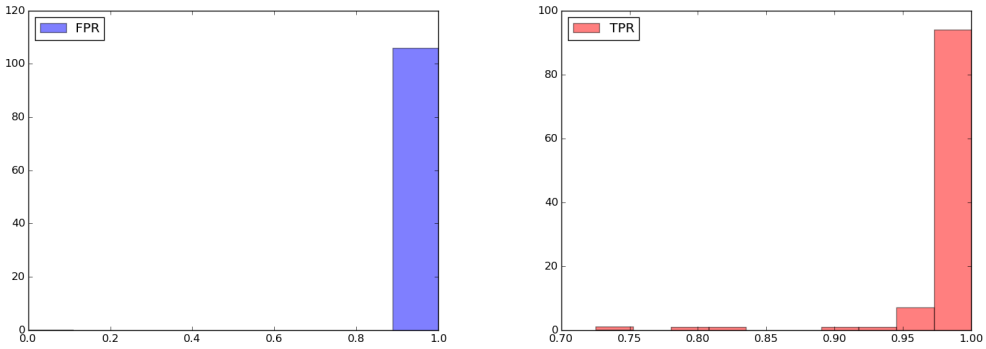


Figure 3: STFT Results

From this baseline, we could conclude that melody (voicing) is not straightforwardly correlated with the predominant frequencies. Technically, only when there is complete quiet (no sound) in the signal period, the STFT will return no voicing detected. This can be proved by Parsevals Theorem [6] as well that the energy (sound) appearing in time domain would cause corresponding energy emerging from frequency domain. The melody, nevertheless, contradicts to this principle completely in that melody is a higher level element in music. The existence of energy in time domain is merely necessary for melody detected, but not sufficient. This essentially closes out the simple and low-level Fourier approaches, leading us to Machine Learning area.

- **SVM on STFT**
  Besides the above two baselines we have explored, now we integrate SVM into previous experiments which trained linear classifiers on STFT features. After extracting STFT for each song, we throw the normalized features into the SVM and predicting voice presentation. The first baseline method was trained using Short Time Fourier Transform (STFT) as feature, which was calculated using 1024-point Discrete Fourier Transform, a 1024-point Hanning window with 205 feature vector length, i.e. extracting every 5th values in the original vector with 1025 length. The reason we want to start with STFT is that there was published result using SVM to analyze the classification problem for melody transcription by Graham E. Poliner and Daniel P.W. Ellis in 2006 [5] which shows comparable results to other methods but with different dataset that we have. The chosen normalization method is one of three approaches mentioned in the paper that was applied within each time frame with mean 0 and unit variance, same normalization criterion was applied on validation and testing sets to ensure the integrity of the data. And we have it as **54.8%** for voicing detection
- **SVM on Raw Audio**
  Another baseline method we have tried is training the model on raw audio waves. Without doing any adjustment, we simply use raw audio as features from the song with 128 feature dimensions for each time frame (4ms). The general results using default learning parameter in the model (C, penalty, loss, dual) were used to train the classifier. Unsurprisingly, using raw audio to predict melody performed slightly better than random guess that for voicing detection task, it has **50.6%** accuracy on the binary classification problem.

8 Approaches

In this section, we discuss our machine learning approaches to melody extraction. We started by examining traditional machine learning systems such as Support Vector Machines and Hidden Markov Models. Later, we moved to deep learning architectures, which we believed to be better suited to learn the high level patterns of melody.

[6] http://en.wikipedia.org/wiki/Parseval%27s_theorem

8.1 Support Vector Machines

As we have stated before, melody extraction can help improve musicology analysis and various signal transformation applications. And voicing detection is part of the process which can be treated as a binary classification problem. The best performing melody extraction methods are signal processing methods, which are rule-based. Here we want to try another machine learning algorithm, Support Vector Machines (SVM), which needs no rule-based analysis to detect the presence of melody.

Generally, SVM is one of the best off-the-shelf supervised learning algorithms that uses linear function to construct a hyperplane that to separate training points into two classes. For example, in two dimensions, we wish to find a hyperplane that can separate the points. Geometrically, it can be visualized as Figure 4:

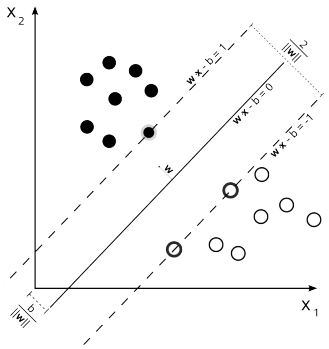


Figure 4: Support Vector Machine Hyperplane

We trained a classifier that can separate the labeled training data in the given space. We assume that each time frame is independent with others and thus we can perform some normalization methods on each data instance (time frame in our case). Typically, SVM involves with kernel trick to map non-linear separable data points into higher dimensions and makes it easier to be separated, though at higher computational cost. Mathematically, we want to solve the following problem:

$$\arg\min_{(w,b)} \frac{1}{2}\|w\|^2$$

subject to for any i = 1, ..., n

$$y_i(\boldsymbol{w}\boldsymbol{x_i} - b) \geq 1$$

For this project, we constructed non-probabilistic binary classifiers to separate points with melody and without melody for stage 1. We then trained a separate model for predicting melody notes (we separated out any time steps with 'no melody' labels). We treated melody note prediction as a regression problem and evaluated its result using R^2, since we want to put our attention on how good our model can fit into the data even with such various background of the data. The training file is originally formatted as a mono audio file (.WAV) and each file was transformed into .csv file as data pre-processing stages for feature extraction. In order to form a good comparison with other methods, our training, validation and testing data kept the same split which is described in the data pre-processing section. Features we used include CQT(84 feature vector length for each frame) and MFCC(20 feature vector).

8.1.1 Experiment

Feature Optimization

We use MFCCs and CQT combined together as newly extracted features for SVM. In total, we have 104 feature dimensions for each one of the 108 songs.

Starting from thinking about the optimization methods that might be useful in our case, we tend to consider the distance between two data points that hugely affect the value, we normalize the

features by two methods: within each dimension, we scale it into [0, 1] or within each time frame, we normalize the data into mean 0 and unit variance. It helped to improve the voicing detection accuracy into 68%.

Kernel and Parameter Tuning

Intuitively, Radial Basis Function (RBF) would be our first choice of kernel for SVM since RBF gives us more flexibility for constructing hyperplane comparing to other kernels. However, the running time for one round takes overnight to get any valuable results. Thus, we perform the combination of PCA and feature importance returned from Random Forest on the data dimension and it reduces the feature space from 104 to 100 with 98% variance captured in the data. The computational time reduces to less than 10 hours but still a huge investment on the unnecessary output with non-significant improvement on model performance (67%). With more time, we could have tried to use the Fourier transformation to approximate RBF kernel values which would help scale better to large numbers of training samples or large numbers of features in the input space. Without further going into dual space, we decided to stick to primal space for analysis and chose the linear kernel as our main kernel for the following SVM exploration.

We used the linear support vector Python implementation LinearSVC [7] in Scikit-Learn as our primary package for SVM as it is implemented in terms of `liblinear` rather than `libsvm` [8] which offers more flexibility of hyperparameters options such as loss functions, penalty, space choices and more applicable to larger sample set.

Due to the sparsity of the feature space, we tried our penalty as lasso for better performance and classic loss function for SVM, but primal space for smaller computation cost. C value varies from 10^{-5} and 10^5 to span the large enough magnitude and once we found out the best range, we "zoomed in" to fine-tune the hyperparameter. Instead of using ridge penalty in traditional SVM, we changed to lasso, 1-norm SVM [10] proposed in 2004 stated that respecting to sparsity principle for high dimensional problem, it encourages for using lasso penalty, which produces a sparse solution in SVM and is thus more robust to noise features and should be closer to the real solution. The setting that we just described yields an accuracy score of 67.9%, though has a faster training time for melody detection as binary classification problem.

Weighted Sample

To weight samples, we integrated Adaboost algorithms into SVM. In Adaboost, for each round, we are allowed to update the weight for points which are misclassified before it goes into the following procedure. Initially, all sample started with weight $\frac{1}{n}$ where n represents the total number of instances in the training set, and using SVM with c values returned from best performance in previous section as a weak learner, to pick out the points that we want to reweight, and train the classifier for N rounds. We have tried the iteration for [50, 100, 300, 500, 1000] and considering the time limitation, when the number of rounds goes beyond 500, it is hard to get any results for a reasonable time commitment. And the best result comes from 300 rounds with 68.6% accuracy score for voicing detection.

Random Forest

Another simply model we constructed is random forest with 50 parallel jobs processing at the same time and 500 trees for the task (determined by cross-validation) which performs similar to SVM with the same features which leads us to think that points are not easy to be separated and as one of the reasons to give up on using RBF kernel, and it also shows that this is not a problem of linear kernel that we have decided to choose.

8.1.2 Results

After we examined the above proposed optimization methods, we summarized the model performance by the following graph. Since MFCCs and CQT is more popular in speaker authentication and more general than STFT, in terms of accuracy score and R^2 for two stages, voicing detection and raw pitch estimation, methods performed on CQT and MFCCs turns out to have much better results than the model using STFT and raw audio. Within the models using normalized CQT and MFCCs,

[7] http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

[8] http://www.csie.ntu.edu.tw/ cjlin/libsvm/

sample being weighed shows slightly more accurate than using equal weights for each sample. One possible explanation could be that samples that are difficult to predict are those points in the margin between melody and 'no melody' switch. Thus, by focusing on classifying those cut-off points and increasing their weights could be better predicted if we look into the data and see which samples' weight are increased through out the tuning process.

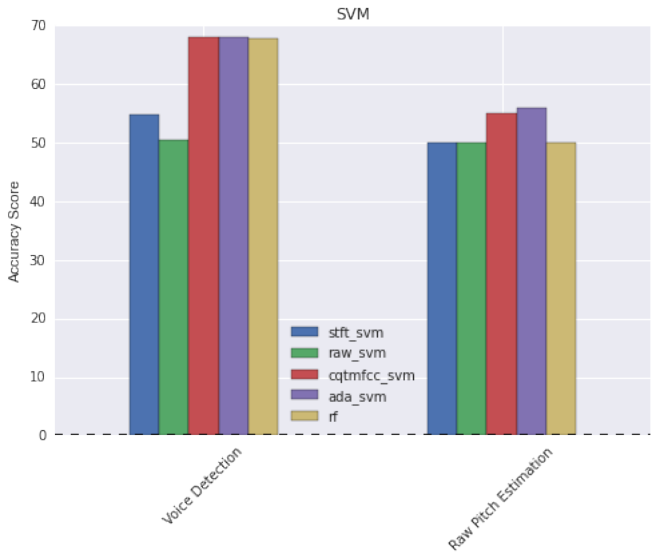


Figure 5: Voice Detection: Blue and Green are baseline models while others are improved models

Methods	Voice Detection Accuracy	Raw Pitch R-sq	overall accuracy
CQTMFCC_SVM	0.679	0.582	0.189
Ada_SVM	0.684	0.591	0.179
Random Forest	0.682	0.573	0.156

Figure 6: SVM results overview

8.1.3 Error Analysis

As we explored the methods, we found some interesting facts that addresses our attention. While tuning the parameter for SVM model, all of the methods seem to insensitive to the C value, which tells SVM how much we want to avoid misclassifying each training point. Comparing $C = 10^5$ with $C = 10^{-5}$, it is pretty much the same across the range, from 66% to 67.9%. It leads us to think about our data set that while it prefers small C values that less regularize the training data and also without producing much validation error, due to the large size of the data, it is less possible to have overfitting or underfitting problem. And by proving this thought, we tried the experiment on randomly selected songs (about 5) and tune the parameter C only. It does return with some valuable findings if we change the C value, i.e. C value influence the performance of the model and prevent overfitting problems. However, the larger the training data, the less it is likely to have this problem. When we have different subsets of training data (5 songs in each, 3 for training, 1 for testing and 1 for validation), it returns with variously prediction result even with the same model for the whole dataset.

Feature changes from STFT/raw audio to CQT and MFCCs, and the performance improves by a large amount (more than 10%). And the dimension reduction didnt reduce it to a much lower level which means that CQT and MFCCs are generally independent from frame to frame thus further support our assumption about the independence of data instances while STFT/raw are more dependent even though they have higher dimension but less able to capture the entire data characteristics.

Another fact that we want to point out is even in raw pitch estimation part, all three models, CQTM-FCC_SVM, Ada_SVM and Random Forest captured more than 50% of the variance in the model when concatenated the stages together and output the overall accuracy, it shows pretty disappointed results which all of them are lower than 20%. One of the reasons that we think is that due to it is a multi-class problem in the whole melody extraction tasks, there are more than 80 class labels which makes it extremely hard to classify the points comparing to regression problem.

8.2 Hidden Markov Models

Given the sequential nature of our data, we decided to try a Hidden Markov Model. We began by using our melody labels to construct a transition matrix, where the labels themselves defined the hidden state space. The no melody class is labeled as class 0. The Figure 7 below is a heat-map of the transition matrix.

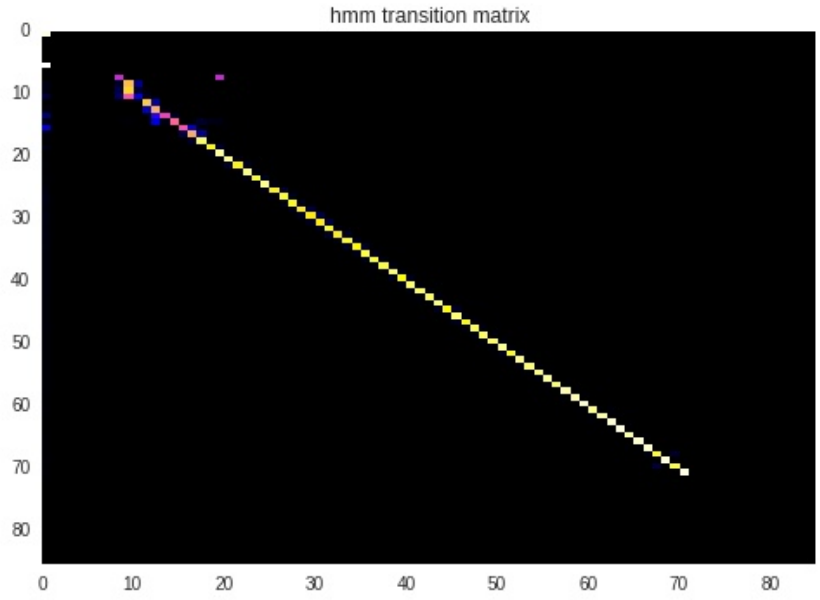


Figure 7: HMM Transition Matrix

It is clear from the figure that the transition matrix provides little insight into the structure of melody. The vast majority of transitions occur from one class back to itself. Thus, we expected simple HMMs to perform quite poorly.

We grouped the CQT transforms at each time step according to melody state and calculated the mean and covariance matrix for each group. These parameters and the transition matrix were fed to a Gaussian Hidden Markov model Viterbi decoder in the Python Scikit-Learn package [9] .

The HMM model achieved 32.5% raw pitch accuracy, 58.2% voicing detection accuracy, and 35.0% overall accuracy. At this point, we decided that, due to the shape of the transition matrix and our general intuition that processing relationships across longer periods of time is critical to melody extraction, that our time would be better spent on deep learning architectures.

8.3 Deep Convolutional Neural Networks on Raw Audio

In light of current success of deep learning approaches from Computer Vision, Speech recognition and Natural Language Processing, end-to-end learning becomes durable and popular. Rather than

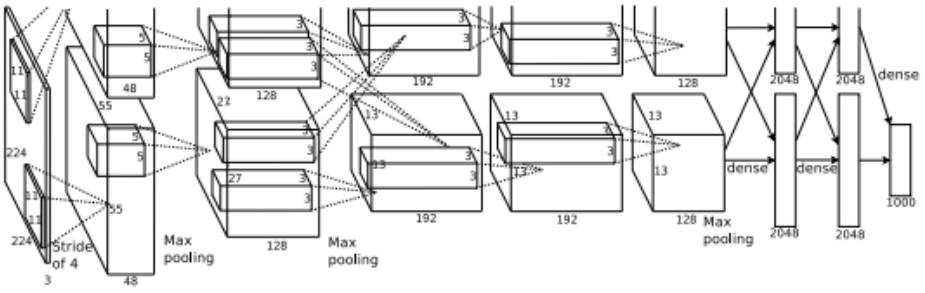
[9] http://scikit-learn.org/stable/modules/hmm.html

learning a classifier or regressor based on hand-crafted features, end-to-end learning advocates us to learn a deep neural network alike framework on raw inputs. For vision and images, that means pixels [19]. Characters and phoneme are used respectively in NLP [20] and Speech [21]. Therefore, we make an attempt to train deep net on raw audios. Thanks to the modern parallel computing GPU chips, the training speed has been boosted by a lot. One can also apply 1-D convolution on raw audio data which further reduces the burden brought by deep net training.

8.3.1 Convolutional Neural Network

Convolutional neural networks (ConvNet) have been proven successful in computer vision, sequence labeling[10] , and music information retrieval tasks. In particular, convolutional neural networks have been shown to be successful in genre classification of music [11]. Thus, we sought to explore whether or not convolutional networks would be successful in melody extraction.

The fundamental Convnet applied on 2D images are shown below [AlexNet]:



Convnet is tenchnically a bottom-up architecture. It is constructed by alternating between convolution, pooling, non-linear activation in the previous few layers and in turn, followed by a couple of fully-connected layers on top. By feeding top layer unit into a multinomial logisitic regression can it get the final classification results.

1. **convolution** Each convolutional kernel is known as a particular pattern detector that only responds to a specific pattern appearing in the input space. The combination of them would form combined feature respondances being explored in the upper convolutional layers.
2. **pooling** Max pooling is regarded as the one of the keys serving the success of current supervised deep learning architectures. It essentially subsamples the input signal by taking out the predominate component within each chunked small window, and uses that component to represent each whole window.
3. **non-linear activation** Rectified linear function, $max(0, x)$ is usually adopted. The reason of using non-linearity in the network is for the following reasons. 1. It gives the network capacity to approximate more complicated distribution. 2. It avoids the consecutive layers from learning trivial results such as Identity. 3. It acts like regularization making the feature planes sparsite, by its definition.
4. **fully-connected layers** The top convolutional layer will be stretched to vector and fed into fully connected layers. The same non-linear activation is performed as well in fully-connected layers. It is followed by a multinomial classifier.

It has been shown that convnet is capable of discovering latent patterns in input data. For example, in speech recognition, the adjacent raw phonemes (elements of spoken words) are highly correlated. Convolution operation essentially takes into consideration this correlation and manages to model them by a weight sharing strategy. Being more specific, each convolution kernel travels through the whole batch of data in order to extract some prefixed pattern appearing in raw audios. Further, the kernel is learnable in Convnet which enables better feature representation from raws.

[10] http://arxiv.org/abs/1502.01710

[11] http://www.iaeng.org/publication/IMECS2010/IMECS2010_pp546550.pdf

8.3.2 Parameter Choosing

It is of vital importance for choosing appropriate meta-parameters of deep Convnet. The meta-parameters mainly include the size of the convolutional kernels, the width of each hidden layers and the size of the fully-connected layers on top. The infinite meta-parameter space almost negates an experimental search for meta-parameters. Rather, we set the meta-parameters inspired by traditional music analysis principles.

The first layer of Convnet is usually taken seriously, in the sense that for the raw input modeling, the first layer is directly facing them. Either an information bottleneck or too broad a information canal doesnt work. We therefore draw lessons from Fourier signal processing theory. That says, facing with sampled signal (all the raw audio stored in modern computer is digital version which sampled from natural analog signal), there is a strict minimum kernel size corresponding to the frequency that one wants to discover, by Shanno-Nyquist theorem. Therefore, in order to recover the lowerA on the piano, with a sample rate as 22050 Hz, we choose the minimum kernel size 196 roughly.

8.3.3 Experiments setup

We setup several Convnet models and all of them have multiple (4 or 5) convolution layers followed by two fully-connected layers operated with Dropout [22], which is regarded as standard Convnet architecture. It differs slightly from the figure above showed in that 1-D convolution/pooling are adopted instead of their 2-D counterparts.

The number of feature planes (width of hidden layer) varies. Convolutions usually runs with zero padding, and some of them are with strides. We take as one training/test sample the float points within 1 second, i.e. 22016 points after being chunked. As Medleydb[2] has a set of annotations per 256 points, each sample gets 172 samples involved. Note this is because the minimum kernel size in first layer is approximately 224 and if we use each small frame (256 points) as one sample fed into Convnet, the kernel size would be thresholded by 256 and usually it should be much smaller than it. Keeping in mind the kernel is used to discover fixed patterns in the data space, we dont expect it to detect patterns in that extremely short frame. This, essentially, serves as an intuitive explanation of the minimum kernel size deduced from signal processing theory. Therefore, in order to use larger kernels, we choose 1 second as the size of one sample. Owing to the fact that there are 172 labels corresponded for each sample, one can use regression on top of the Convnet whose target is a 172d vector and trained with Euclidean distance, dubbed as step_0. One can also use the label located at halfway of the current second as the label, i.e. step_1. Weve also tried voicing detection by same architecture, i.e. step_2.

8.3.4 Results Analysis

Voicing Detection (step_2) We first focus on voicing detection. The configurations of the model are:

model name	number of feature planes	kernel size	pooling size	voicing detection accuracy (validation)
model1	128-128-128-256-256	515-127-63-63-63	16-14-8-2-2	66.2%
model2	128-128-128-256-256	4101-127-63-63-63	16-7-8-2-2	TBA
model3	1024-256-256-256-256	515-127-127-127-127	16-14-8-2-2	Not converging

Figure 8: Voicing Detection Results

As we can see in Figure 8 above, the $\text{step}_2.\text{model}_1$ serves as a baseline in this series of experiments. By contrast, we tried both increasing the first layer feature plane and first layer kernel size. As demonstrated earlier, an appropriate setting of first layer of deep learning architecture is indispensable for success training. Neither too much information fed upward nor too few works in that the former would give rise to overfitting and latter is subjected to underfitting. As illustrated in Figure 8, the smallest model performs nearly as well as SVM. Increasing the kernel size does help a bit whereas increasing the feature planes doesnt have much effect but increasing the difficulty of training.

Delving into $\text{step}_2.\text{model}_2$, increasing the first layer feature planes intuitively corresponds to the complete Fourier basis from signal processing theory. Both of them tries to factorize the input signal by linear combination of multiple harmonic waves. However, from Machine learning perspective, this would create a 'bottleneck' framework. As we know, bottleneck structure essentially obstacles information going through as one of the causes in which higher layer takes a long while learning to be correlated to lower layer. One possible solution is to remove the bottleneck by increasing the feature planes through the whole network construction, which will hit the limitation of memory of modern GPUs (Since we are using GTX 580 which only has 3GB memory). As a result, model_2 doesn't work unfortunately.

Pitch Tracking (step_0 / step_1) We now move on to see pitch tracking. The configuration of the models are shown in Table 9.

model name	number of feature planes	kernel size	pooling size	learning rate	optimization result
model1	128-256-256 -512	255-127-127 -95	43-8-8-2	0.1 0.05 0.01 0.005 0.001	Diverge Saddle pt Saddle pt Saddle pt Saddle pt
model2	128-128-128 -256-256	197-127-127 -127-127	43-8-4-2-2	0.1 0.05 0.01 0.005 0.001	Diverge Diverge Saddle pt Saddle pt Saddle pt

Figure 9: Pitch Tracking Results

We found that doing a narrow L2 target training is indeed difficult in terms of optimization. To the best of our knowledge, L2 loss is only applied in Auto-encoder, where L2 is obtained from the identical size as input. On the contrary, our models narrow it down to a tiny vector. Thus we tried different learning rate and observe the optimization behavior, the result is shown in Table J2.

Unfortunately, limited by computing resources and time, we didnt manage to get pitch tracking to work by feeding raw audio into Convnet. More exploration would be stressed on the loss function we chose. We could learn from the failure of this pipeline of experiments that fitting the Convnet backproped by a narrow L2 loss seems not working in terms of the touch optimization. It is widely assumed optimization is one of the keys of the success of deep learning, though this process is always intractable and not manageable in its highly non-convexity surface lying in extremely high dimensional space.

In a word, fitting a deep Convnet with a regression L2 loss on top makes trouble for optimization, which signifies the difficulty of using deep learning technique on raw audio to learn a pitch tracking system.

8.3.5 Explanation

End-to-end learning becomes possible in computer vision, speech and NLP. However, the understanding of music differs and we conjecture that more labeled data is in demand. We will resort to adopting Convnet on off-the-shell engineering features demonstrated later and we briefly conclude this section that end-to-end learning on music needs further exploration.

8.4 Deep Convolutional Neural Networks on CQT

As discussed in section 6, the CQT transformation provides the energy at each frequency across time. Concretely, the CQT transformation that we used is a matrix with 84 rows across the frequency domain, and thousands of columns across time, where the number of columns varies with each song.

In image recognition tasks, convolutional neural networks have been shown to be extremely effective at detecting and classifying complex patterns from pictures. In a sense, the CQT transform of a song can be viewed as a series of CQT images across time. Thus, we decided to view the task of identifying melody from CQT transforms as an image recognition task.

In a typical song, there are usually at least a few types of instruments playing, such as drums, guitar, bass, saxophone, singing, etc. The melody is defined as the fundamental frequency of the predominant instrument, and can often have a more varied pattern along the frequency domain. In other words, it can go up and down more than other sound patterns one might see in the CQT. For example, in the figure below, the singers voice is the melody, and tends to move around more along the frequency axis (though this is not always the case). Meanwhile, the guitar strumming in the background can be seen as the long, flat horizontal white lines. The drums are lower in frequency, and look like big blobs along the bottom of the chart. Sometimes, there is no melody playing.

We used a convolutional kernel that spanned both the time and frequency domains. There are a couple of reasons for this. We would expect that there are certain kinds of melodic patterns that will tend to show up across different songs. We would also expect that the same melodic patterns will appear at different octaves or along different melodic scales. In other words, there should be time-invariant and frequency-invariant representations of melody across songs, which make the convolutional neural network seem to be a good choice for this problem.

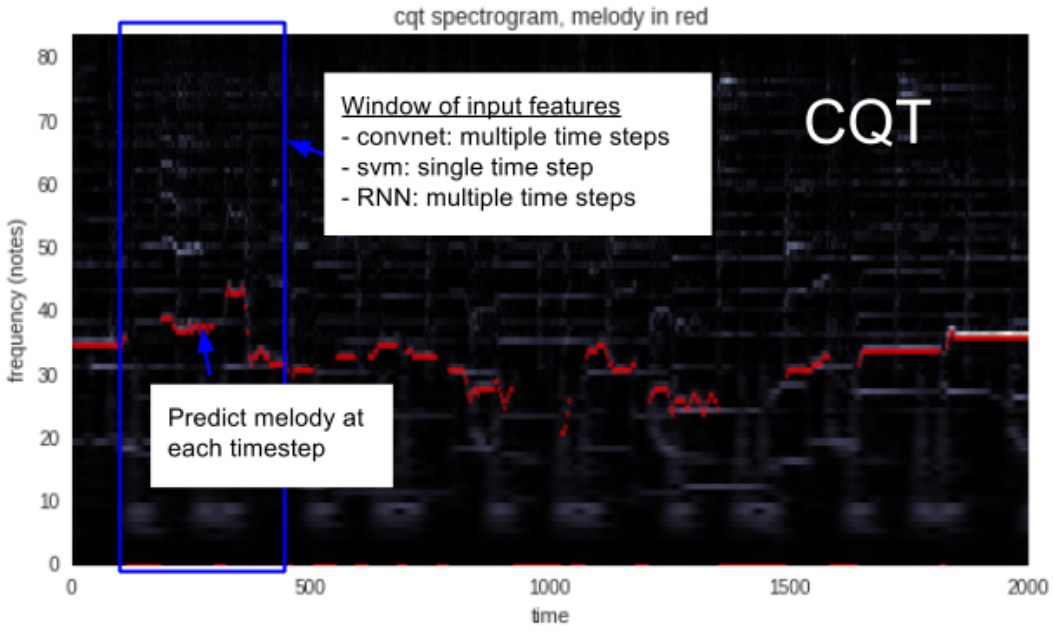


Figure 10: CQT Spectrum

We first employed the convolutional neural network to the task of voicing detection, which we presumed to be a much simpler task than pitch tracking. It seems reasonable to assume that if a model does not perform well on voicing detection, it will not perform well on pitch tracking. We learned later that this may not be true.

Hyperparameter tuning for convolutional neural networks is not straightforward, as there are many parameters to choose from, and performing a grid search across all options is simply not feasible. For example, there are the number of layers, size of each layer, types of nonlinearities, dropout [12][12], etc. In addition, for time series data, the size of the time series window is yet another parameter. A general strategy we use here is to use a relatively small network (because larger networks take longer to train) and to focus first on those parameters which we believe will yield the largest change in model performance, and adjust only one parameter at a time. Once these hyperparameters have been tuned, we can then vary network size.

We focus first on convolutional filter size. It is typical in image recognition tasks to use 5x5 convolutional filters, or something similar in size such as 3x3 or 7x7. Thus, the first test uses a 5x5 filter.

The CQT spectrum across time, however, is not like a typical image. When an instrument plays a note, such as a guitar playing C2, a number of energies will appear across the frequency domain, beginning at the fundamental frequency of C2. The energies above the fundamental frequency are known as overtones. The image below is a CQT transform of a female singer audio clip over a few seconds.

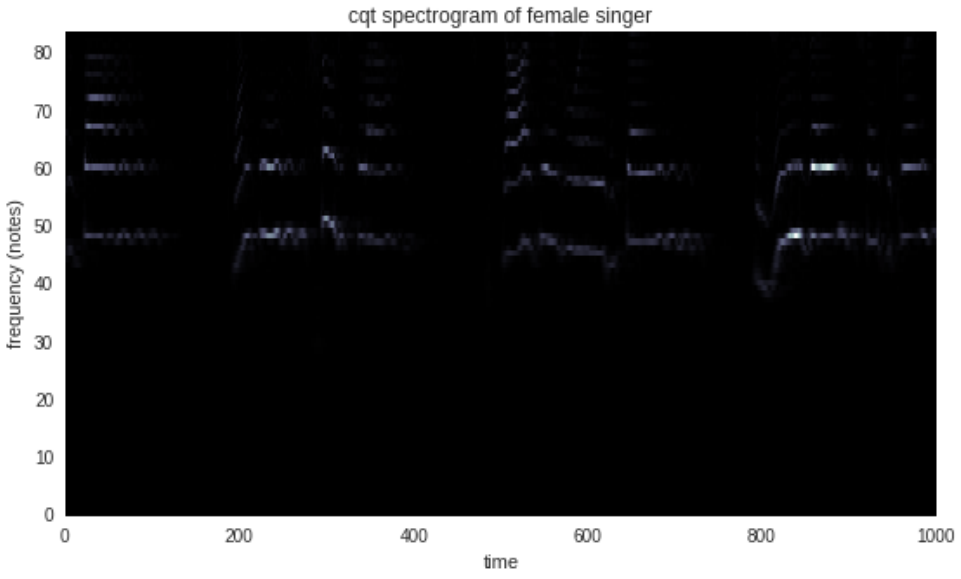


Figure 11: CQT Spectrum of Female Singer

Notice that the energy patterns begin at a note of around 40-50 and seem to be replicated at higher notes. This type of pattern is common for all instruments that we are likely to see in our dataset. Thus it stands to reason that we might want a larger convolutional filter size that encompasses the majority, or even entirety, of the length of the overtones. In addition to 5x5 filters, we test filter sizes that span 21 and 41 notes in the frequency domain.

The second hyperparameter tuning is the size of the time window. A larger time window yields more information for each prediction, and thus should yield improved performance. If the window is too large, there may be so many irrelevant features that performance begins to decrease.

[12]Dropout is a regularization technique. It is analogous to bagging and has been shown to be more effective than parameter shrinkage.

We also experiment with normalization, which we would expect to improve performance. An additional hyperparameter we did not experiment with, though would have liked to with more time, was the width of the filter kernel (we only varied the height).

For the first ten experiments we keep the network depth and width fixed with two convolutional layers with width 64 and 128 feature maps, respectively, and two fully connected layers with 256 and 32 hidden nodes, respectively. The second convolutional layer uses a 2x2 max pooling subsampling layer, and all layers use rectified linear units (ReLU) for the nonlinearity. All experiments use 70% dropout on the fully connected layers. All models were trained in Torch7 [13] with backpropogation for 40 epochs using ADADELTA [13], which is an adaptive learning method for gradient descent, with momentum 0.99, and a randomized mini-batch sample size of 100.

The results of the experiments are shown in the Figure 12 below. In the first three experiments, it seems that increasing the filter height to 41 improves the accuracy of the model. Thus, in experiments 4-9, we fix the filter height to 41 and vary the time window from 20 seconds (i.e. 10/10) up to 70 seconds (i.e. 35/35). There does seem to be an optimum time window at around 40 seconds (i.e. 20/20), with an accuracy improvement of 3%. Given this result, we fix the time window at 40 for experiment 9 in which we test normalization. This does not yield any measurable difference in performance. In experiments 10-12, we increase the depth of the network and vary the width of the network, but this yields no improvement in performance.

Figure 12: Voicing detection using convolutional neural networks

#	Architecture	Dropout	Time Window (Left/Right)	Normalization	Validation Accuracy
1	64c5x5_128c6x5P_256f_32f	70%	10/10	None	67.2%
2	64c5x21_128c6x5P_256f_32f	70%	10/10	None	67.9%
3	64c5x41_128c6x5P_256f_32f	70%	10/10	None	68.8%
4	64c5x41_128c6x5P_256f_32f	70%	15/15	None	70.4%
5	**64c5x41_128c6x5P_256f_32f**	**70%**	**20/20**	**None**	**72.1%**
6	64c5x41_128c6x5P_256f_32f	70%	25/25	None	71.9%
7	64c5x41_128c6x5P_256f_32f	70%	30/30	None	70.7%
8	64c5x41_128c6x5P_256f_32f	70%	35/35	None	71.3%
9	64c5x41_128c6x5P_256f_32f	70%	20/20	Mean, Std	71.7%
10	64c5x41_64c6x5P_64c5x5P_256f_32f	70%	20/20	None	71.7%
11	64c5x41_128c6x5P_128c5x5P_256f_32f	70%	20/20	None	70.7%
12	64c5x41_128c6x5P_256c5x5P_256f_32f	70%	20/20	None	71.1%

**Note: The architecture key is as follows: 64c5x41P indicates a convolutional layer, denoted by c, with 64 feature maps, a convolutional filter with width 5 (time domain) and height 41 (frequency domain), and a max pooling subsampling layer; 256f indicates a fully connected layer with 256 hidden nodes.*

These results were slightly disappointing, given that they are only slightly better than using an SVM. We would have expected much higher performance, which might hint that we do not have enough data for the application of convolutional networks.

[13] http://torch.ch/

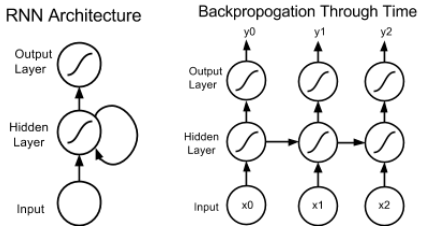
Based on these experiments, it seems that hyperparameter tuning does not yield much improvement in performance. Further, it seems that increasing the depth and width of the network also does not provide much boost in performance. Thus, we can reasonably conclude that we are likely not going to see much improvement in performance by continuing to tune hyperparameters.

Next, we focused on the full melody extraction task, in other words pitch tracking + voice detection. Unfortunately, we were unable to successfully train a convolutional neural network for this task. The network predicted no melody in all time steps of all epochs. This was likely a training issue rather than a model issue. However, we tested different training schemes, such as training over entire songs vs training on mini-batches of samples from different songs, as well as using SGD vs ADADELTA. None of these attempts seemed to help. At this point in the project, we decided to move on to recurrent neural networks.

8.5 Deep LSTM Recurrent Neural Networks

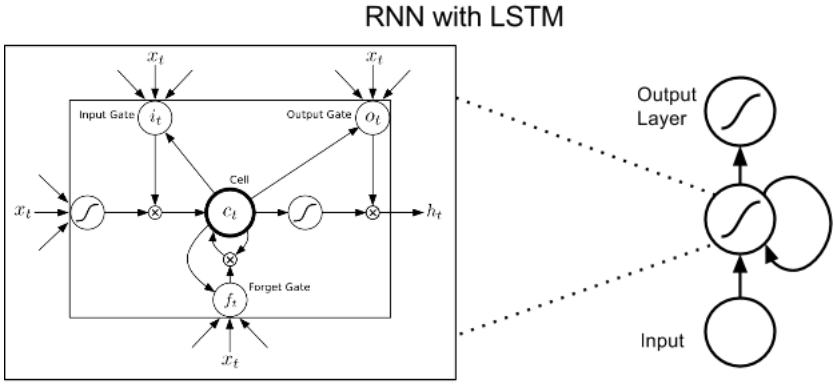
Recurrent neural networks (RNN) with Long Short-Term Memory (LSTM) cells have been successfully applied to many sequence-based tasks, such as speech recognition and handwriting generation [14], music generation [15], and character and word sequence labeling [16], amongst others. We are not aware of any examples of application of LSTM applied to melody extraction. Given that LSTM cells are known for being suited to modeling long-term dependencies in sequence-based tasks, we expected that it would perform well in melody extraction.

An RNN is a neural network with cyclic connections. For example, a hidden layer's outputs could be one of its inputs, which is the type of architecture we use in this project.



RNNs can be trained with backpropogation through time (BPTT). BPTT unfolds the network into a discrete series networks connected in sequence and trains it as if it were one giant acyclic neural network.

Long short-term memory cells are a special type of RNN architecture that are known for reducing the vanishing and exploding gradient problems of RNNs and and better at exploiting long-range structure in sequential data. The LSTM cell [17] replaces the nodes of the hidden layers in the RNN. For a more detailed explanation of RNNs, LSTM, and BPTT, see [16].



source: http://www.cs.toronto.edu/~graves/preprint.pdf

LSTM on Voicing Detection

As with the convolutional neural networks, we first deployed LSTMs to voicing detection, since we presumed this to be a much easier task. The hyperparameter selection strategy was to fix the network size to be relatively small and to focus first on those parameters which we believed would yield the largest change in model performance, and adjust only one parameter at a time. Once these hyperparameters had been tuned, we could then train on a larger network. Figure 13 lists the results of the experiments. All modeling experiments were constructed and performed in Torch7. RNN and LSTM architectures were constructed using the nngraph package. All models utilized a negative log likelihood objective function with log softmax as inputs. All models were trained out to 50 epochs with no dropout. We experiment with dropout in later melody extraction experiments.

Figure 13: Voicing detection using LSTM networks

#	Architecture	Training Method	Max Norm	Dropout	Time Window	Normalization	Validation Accuracy
1	1l_200n	SGD	2	0%	20	Mean, Std	68.7%
2	1l_200n	SGD	2	0%	40	Mean, Std	69.7%
3	1lp_200n	SGD	2	0%	20	Mean, Std	69.5%
4	1lp_200n	ADADELTA	2	0%	20	Mean, Std	69.3%
5	1l_200n	SGD	5	0%	20	Mean, Std	69.0%
6	1l_100n	SGD	3	0%	20	Mean, Std	69.5%
7	1l_400n	SGD	3	0%	20	Mean, Std	70.2%
8	2l_200n	SGD	3	0%	20	Mean, Std	69.1%
9	2l_400n	SGD	3	0%	20	Mean, Std	69.4%
10	3l_200n	SGD	3	0%	20	Mean, Std	50.7%
11	4l_200n	SGD	3	0%	20	Mean, Std	50.7%
12	**2l_200n**	**ADADELTA**	**3**	**0%**	**20**	**Mean, Std**	**71.7%**
13	3l_200n	ADADELTA	3	0%	20	Mean, Std	70.9%

**Note: The architecture key is as follows: 1l_200n indicates an RNN with 1 hidden layer with 200 LSTM cells. 1lp_200n indicates the same architecture, but with peephole connections.*

We first focused on the size of the BPTT sequence in experiments 1 and 2. A larger BPTT sequence, in other words a larger time window, would allow the network to learn longer range dependencies. We tested BPTT sequence lengths of 20 and 40 timesteps. It turned out that 40 timesteps yielded a 1% performance improvement on the validation set, though this was not significant enough to be of major interest.

Next, in experiment 3, we turned on peephole [18] connections, which are connections that can be added to the LSTM cell between the cell value and the input, output, and forget gates. Gers et al. 2002 showed that these additions can improve the performance of LSTM architectures. However, in our experiments, it did not seem that the peephole connections improved performance, and if anything decreased performance.

Knowing how to effectively train an RNN was another unknown to the team, and so we also evaluated the effectiveness of SGD versus ADADELTA. In experiment 4, we tested ADADELTA on the peephole connection model from experiment 3. The performance difference was largely negligible.

In BPTT, there is an additional hyperparameter which constrains norm of the overall gradient at each iteration to be less than or equal to some value which is set by the user. If the calculated gradient turns out to be larger, then all gradients are shrunk to meet this max norm cap. This technique prevents the exploding gradient problem, and should be thought of as a training parameter rather than an architecture parameter. We adjusted this in experiment 5 from 2 to 5, with negligible change in model performance.

Next, in experiments 6-10, we adjusted the depth and width of the network. In these experiments, the best result was obtained using a wide 1 layer network with 400 LSTM cells. Deep networks with 3 and 4 layers were unable to converge to a solution.

It was surprising to us that a wide 1 layer network would yield better performance, given that Graves et al. 2013 found that using a deeper network tended to work better in sequential tasks. Thus, in experiments 12 and 13, we repeated the 2 and 3 layer network experiments with ADADELTA, which is better at training out of overfitting holes. These experiments yielded the best results with 71.7% accuracy on 2 layers.

We did not experiment with dropout, as this was an additional hyperparameter tuning step that was saved for melody extraction experiments.

Overall, the hyperparameter tuning improved voicing detection accuracy by 3%, which was rather disappointing, and led us to conclude that it was unlikely that further tuning would yield significantly better results. We chose not perform an in-depth error analysis for the voicing detection task, and instead left that for melody extraction, which is our ultimate goal.

LSTM on Melody Extraction

Next we tested the LSTM networks on melody extraction (voicing detection + pitch tracking). We model this as a multi-class classification problem with 85 classes (84 notes + 'no melody') using CQT as input (normalized within each song). Note that we contained no labels for some of the classes (very low and very high notes). This should not hurt model performance and makes organization of inputs and outputs easier.

The results of the experiments are shown in Figure 14. All networks were trained to 50 epochs with a BPTT sequence of 20, ADADELTA, momentum 0.99, and max gradient norm cap of 3.

Figure 14: Melody extraction using LSTM networks

#	Architecture/ Dropout	Training/ Max Norm	VxR	VxFP	Raw Pitch Accuracy	Overall Accuracy
1	2l_200n / 0%	ADADELTA / 3	54.9%	17.2%	34.7%	58.9%
2	3l_200n / 0%	ADADELTA / 3	62.3%	23.1%	38.2%	57.8%

**Note: architecture key is as follows: 1l_200n indicates an RNN with 1 hidden layer with 200 LSTM cells. 1lp_200n indicates the same architecture, but with peephole connections.*

In our first two experiments, we tested both the 2 layer and 3 layer networks with zero dropout. In contrast to the voicing detection task, for melody extraction we focus on three additional evaluation metrics: voicing recall VxR (i.e. true positive rate of voicing detection), voicing false alarm VxFP (i.e. false positive rate of voicing detection), raw pitch accuracy (i.e. accuracy of pitch when melody is present), and overall accuracy. These are commonly used metrics used in the melody extraction community.

Our primary observation of experiments 1 and 2 was that the raw pitch accuracy was quite low at around 34%. This is considerably better than random guessing (7%), but not not much better than the HMM, and overall worse than we had expected. Thus we performed an error analysis of experiment 1, the results of which are shown in figures 15 and 16. It seems that the largest issue with the model is that it has great difficulty in predicting that a melody is actually present, as most of the error is attributable to false negatives (i.e. predicting that there is no melody present when

the converse is true). Figure 15 below shows this for all notes. Some bars are missing where we had no examples of that melody note in our training or validation set. Figure 16 shows errors across all notes. Almost 50% of all errors are attributable to false negatives. Roughly 20% of errors are incorrect patch labels. And roughly 30% are accurate predictions. Thus, when a melody is actually detected, the model has approximately 60% accuracy.

Figure 15: Melody Extraction Error Analysis

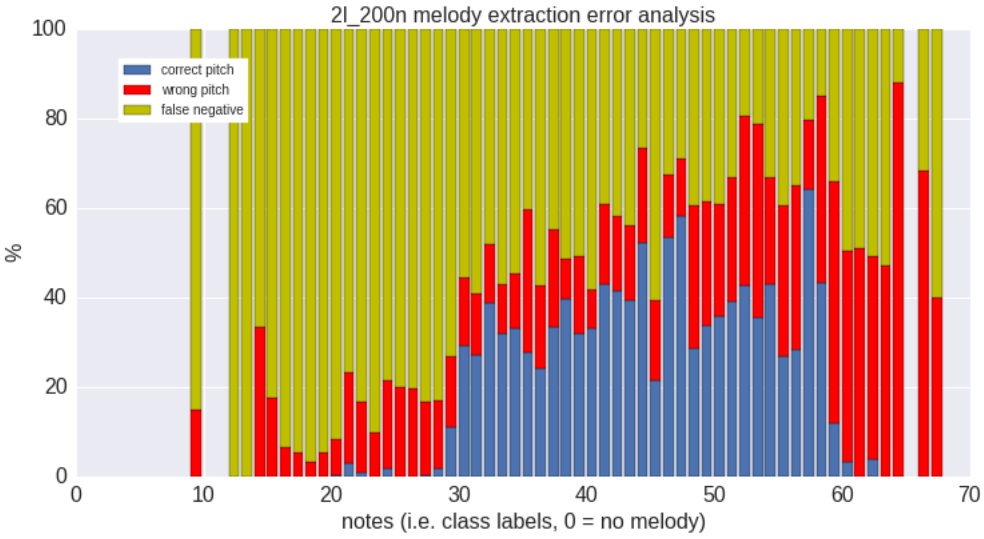
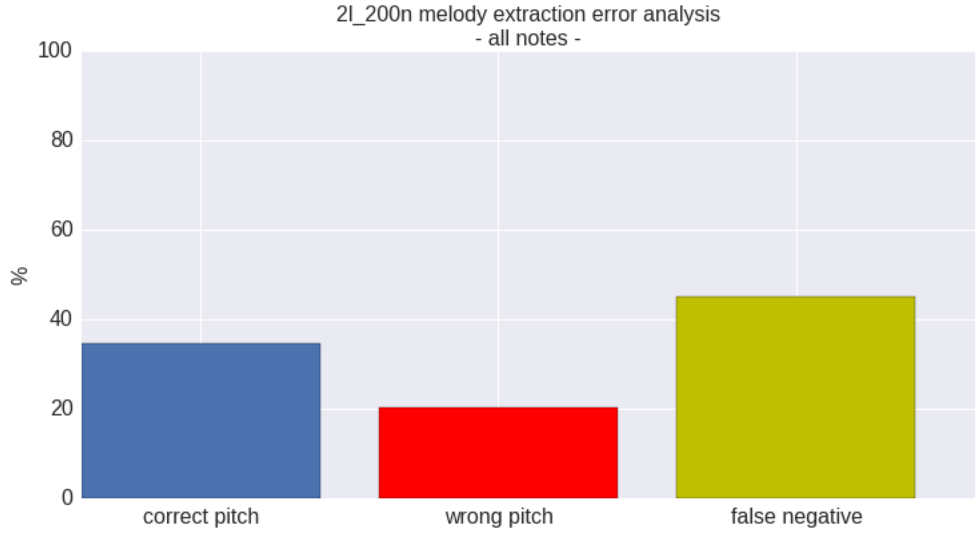


Figure 16: Melody Extraction Error Analysis All Notes



We suspect a couple of reasons for these errors. It is possible that the imbalance in class labels is to blame for these results. Approximately 55% of labels are class no melody, while the majority pitch is only 7%. To test this possibility, we applied weights to the negative log likelihood function. In other words:

$$\sum_{i=1}^{n} \ell(x_i, y_i)$$

becomes

$$\sum_{i=1}^{n} \alpha_{y_i} \ell(x_i, y_i)$$

where α_{y_i} is the weight applied to the loss on predictions of class y_i.

We used a simple weighting schome where only the weight on class no melody was adjusted by weights 0.05, 0.10, etc. All other weights were left at 1. With this weighting scheme, we would expect the model to place significantly less emphasis on incorrectly predicting melody when in fact there is none, and thus the model should be better at detecting the presence of melody. The table below shows the results of these experiments. All networks were trained to 50 epochs with a BPTT sequence of 20, ADADELTA, momentum 0.99, and max gradient norm cap of 3.

Figure 17: Results 1

#	Architecture/ Dropout	Weight for 'no melody'	VDA	VxR	VxFP	Raw Pitch Accuracy	Overall Accuracy
1	2l_200n / 0%	0.01	65.4%	91.7%	60.5%	50.1%	44.8%
2	2l_200n / 0%	0.025	69.9%	81.6%	41.6%	48.0%	53.2%
3	2l_200n / 0%	0.05	69.4%	63.7%	25.0%	37.8%	56.6%
4	2l_200n / 0%	0.10	69.2%	54.0%	15.7%	33.3%	59.1%
5	2l_200n / 0%	0.25	68.6%	51.0%	16.4%	33.1%	58.6%
6	2l_200n / 0%	0.50	67.4%	57.3%	18.8%	36.0%	58.8%
7	2l_200n / 0%	0.75	69.4%	57.3%	18.8%	36.0%	58.8%

It is clear that dampening the effect of incorrectly predicting that there is a melody significantly increases the performance of raw pitch accuracy, though it results in a decrease in overall accuracy. The 0.025 weight case results in a small drop in overall accuracy, but with a large increase in raw pitch accuracy. In all cases, voicing detection accuracy remained roughly constant, except for the 0.01 case, which was heavily biased towards predicting melody.

Finally, we tested dropout on the 2 layer network with 0.025 weighting. The table below shows the results of dropout experiments.

Figure 18: Results 2

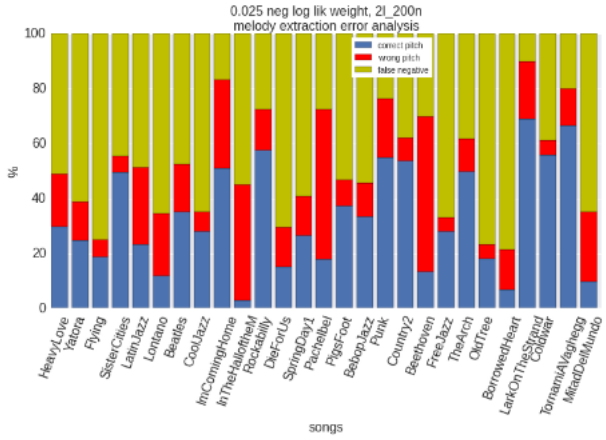
#	Architecture/ Dropout	Weight for 'no melody'	VDA	VxR	VxFP	Raw Pitch Accuracy	Overall Accuracy
1	2l_200n / 10%	0.025	65.4%	83.1%	45.0%	48.5%	51.8%
2	2l_200n / 20%	0.025	69.9%	81.8%	41.4%	45.7%	52.2%
3	2l_200n / 30%	0.025	69.4%	85.2%	47,7%	48.2%	50.2%
4	2l_200n / 40%	0.025	69.2%	84.4%	48.7%	46.9%	49.1%
5	2l_200n / 50%	0.025	68.6%	87.2%	51.8%	50.8%	49.5%

We did not consider these results to be an overall improvement in results, given that the increase in raw pitch accuracy is roughly equivalent to the decrease in overall accuracy. It seems that dropout is serving to further decrease the bias in incorrectly predicting melody when none exists.

One major disappointment, and curiosity, in from these results is that the overall voicing detection accuracy is not better than that for SVM and convolutional neural networks. We are unsure why this would be. Given that humans constructed the melody labels, it is plausible that the upper limit on voicing detection accuracy should be relatively higher. These results may suggests a couple

possibilities. The CQT features may be inadequate, and perhaps we should have made use of the MFCC features. It is also possible that the data set is insufficient.

Figure 19: Melody Extraction Error Analysis

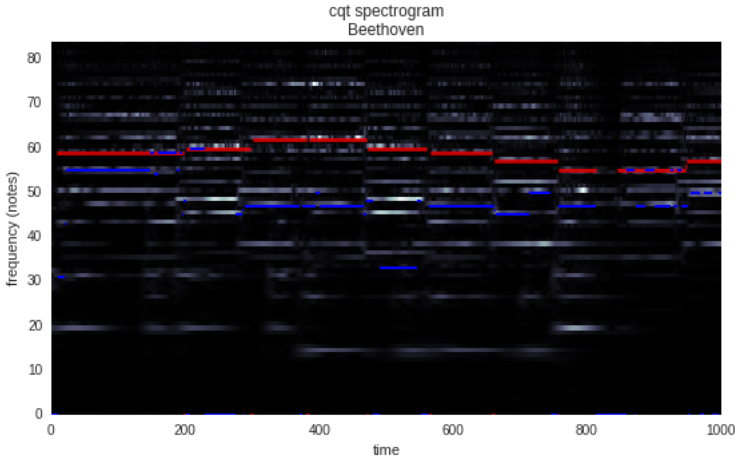


8.6 Insufficient Data

We suspect that there is simply not enough data for this LSTM architecture. Consider the figure below. It shows that the model does well on certain songs, but not others. For example, it achieves around 60% raw pitch accuracy for Lark on the Strand Drummond Castle, Rockabilly, Coldwar, Punk, Tornami A Vagheggiar, which is a diverse set of songs. It is also at around 80% accuracy for detecting the presence of melody in these songs. These results would seem to suggest that there are patterns in these songs that are also found in the training set.

As another example, consider the predictions for Beethoven. The model achieved around 15% raw pitch accuracy on this song. We can look at the CQT with both actual and predicted melody overlaid, which are shown in the figure below. It seems that the model is picking up the general patterns in the melody, but not quite getting the octave right. The melody for this song is quite high in the frequency domain relative to other songs, and we have few training examples with melodies this high. This is likely a primary reason why the model performed so poorly on this song. This finding further supports the hypothesis that we have insufficient data for the LSTM models.

Figure 20: CQT on Beethoven, actual melody in red, predicted melody in blue (on top of red)



8.7 Testing on Final Model

For the final model we chose the 2 layer RNN network with 200 LSTM cells per hidden layer, 0.025 weighting on the no melody class, trained with a BPTT sequence size of 20, ADADELTA with momentum 0.99, and no dropout. We ran this model on our test set. We compare our results against those of Melodia, which is the only other algorithm we are aware of that has been tested on MedleyDB. Given that Melodia established the state-of-the-art in MIREX09 melody extraction from 2011-2014, we feel it is representative of the state-of-the-art on MedleyDB

We should note that this is not exactly an apples-to-apples comparison. Melodia is based on signal processing, and does not have training, validation, and test splits, whereas our method does. Thus we are comparing answers on different datasets within MedleyDB.

Figure 21: Test Result

Method	VxR	VxFP	Raw Pitch Accuracy	Overall Accuracy
LSTM 2l_200n, 0.025 weighting, BPTT 20	82.1%	40.6%	54.7%	57.2%
Melodia, 0.2 nu	78%	38%	55%	54%
Melodia, -1 nu	57%	20%	52%	57%

These results show that our method is competitive with the state-of-the-art.

9 Conclusion

We have considered using machine learning and deep learning algorithms in this project to extract melody of songs. We illustrate this process by constructing SVM, convolutional neural networks, and RNNs with LSTM cells, which all beat the baseline models. The best model for voicing detection was a convolutional neural network, while the best model for overall melody extraction was an RNN with LSTM cells, which produced results comparable to state of the art methods. We believe there is much improvement to be made on these methods. We suggest future improvements in the next section.

10 Future Work

We identified a few key issues with the LSTM model. We suspect that the lack of data was the greatest driver of error in model performance. Thus, leveraging the current data set to generate more training samples would be a fruitful exercise. This is known as data augmentation, the purpose of which is help the model learn to be robust to shifts in the data.

We did not make use of the MedleyDB stems, though these could be used to create new versions of each song, with and without melody. If possible, it could also be useful to inject new computer-generated instruments into songs, though the newly generated songs would need to be sensible. If there is a way to adjust the pitch and speed of the songs in sensible way, such as simply shifting the CQT along the frequency axis, we suspect that these could also be useful data augmentations.

Another issue with melody is that a melody can be the same general melody, but sung at a different speed. For example, try humming a tune such that each note is held for the same duration. It is likely different than the original tune, but it is generally the same. This was an issue not addressed in this project, but should be considered in the future.

With more time we would have tried stacking LSTM layers on top of a convolutional neural network. The intuition here is that the convolutional networks would learn short-term patterns while LSTMs would learn longer term relationships.

In addition, the LSTM networks that we constructed were sequential in only one direction. Given that melody seems to be context dependent in both the past and future, where context can include future time steps, we believe that a bidirectional LSTM network [16] would have yielded improved performance.

Another machine learning approach that we would love to explore is unsupervised learning algorithms given that all our previous work is based on supervised learning. We believe that there might be latent shared similarities and structure between time frames which has melody and those don't have. Using unsupervised learning method, we might be able to simulate the distribution of notes or its density estimation.

As the future work in SVM, we would also love to combine Conv-nets and SVM together as for generic object categorization. In general, the feature learned from convolutional network can be fit into SVM with RBF kernel for large-scale learning task. It comes from the intuition that Conv-nets is good at learning features while SVM is a particular popular toolbox for classification[11].

Acknowledgement

We would first like to acknowledge Aditya Ramesh for allowing us to use his Torch7 optimization utilities repository, which contained the ADADELTA optimization routines used throughout this project. Thank you to Kurt Miller for providing advice and feedback on our project. Thank you to Professor David Rosenberg for a great semester!

References

[1] J. Salamon, E. Gomez, D. P. W. Ellis and G. Richard, "Melody Extraction from Polyphonic Music Signals: Approaches, Applications and Challenges", IEEE Signal Processing Magazine, 31(2):118-134, Mar. 2014.

[2] R. Bittner, J. Salamon, M. Tierney, M. Mauch, C. Cannam and J. P. Bello, "MedleyDB: A Multitrack Dataset for Annotation-Intensive MIR Research", in 15th International Society for Music Information Retrieval Conference, Taipei, Taiwan, Oct. 2014.

[3] J. Salamon and E. Gmez, "Melody Extraction from Polyphonic Music Signals using Pitch Contour Characteristics", IEEE Transactions on Audio, Speech and Language Processing, 20(6):1759-1770, Aug. 2012.

[4] V. Rao and P. Rao, Vocal melody extraction in the presence of pitched accompaniment in polyphonic music, IEEE Trans. Audio, Speech, Lang. Processing, vol. 18, no. 8, pp. 21452154, Nov. 2010.

[5] G. Poliner and D. Ellis, A classification approach to melody transcription, in Proc. 6th Int. Conf. Music Information Retrieval, London, Sept. 2005, pp. 161166.

[6] M. Ryynnen and A. Klapuri, Automatic transcription of melody, bass line, and chords in polyphonic music, Comput. Music J., vol. 32, no. 3, pp. 7286, 2008.

[7] Masataka Goto, Hiroki Hashiguchi, Takuichi Nishimura, and Ryuichi Oka: RWC Music Database: Popular, Classical, and Jazz Music Databases, Proceedings of the 3rd International Conference on Music Information Retrieval (ISMIR 2002), pp.287-288, October 2002.

[8] T.-C. Yeh, M.-J. Wu, J.-S. Jang, W.-L. Chang, and I.-B. Liao, A hybrid approach to singing pitch extraction based on trend estimation and hidden Markov models, in IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP), Kyoto, Japan, Mar. 2012, pp. 457460.

[9] Aharon, Michal and Elad, Michael and Bruckstein, Alfred, "K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation," in IEEE Transactions on Signal Procesing 2006, Vol 54 No 11 pp. 4311.

[10] Zhu J., Rosset S., Hastie S. and Tibshirani R. (2004), "1-norm Support Vector Machines", pp. 3

[11] Huang F., LeCun Y. (2006), "Large-scale Learning with SVM and Convolutional Nets for Generic Object Categorization", pp.1

[12] Hinton, Geoffrey E., et al. "Improving neural networks by preventing co-adaptation of feature detectors." arXiv preprint arXiv:1207.0580 (2012).

[13] Zeiler, Matthew D. "ADADELTA: an adaptive learning rate method." arXiv preprint arXiv:1212.5701 (2012).

[14] Graves, Alex. "Generating sequences with recurrent neural networks." arXiv preprint arXiv:1308.0850 (2013).

[15] Eck, Douglas, and Juergen Schmidhuber. "A first look at music composition using lstm recurrent neural networks." Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale (2002).

[16] Graves, Alex. Supervised sequence labelling with recurrent neural networks. Vol. 385. Heidelberg: Springer, 2012.

[17] Hochreiter, Sepp, and Jrgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[18] Gers, Felix A., Nicol N. Schraudolph, and Jrgen Schmidhuber. "Learning precise timing with LSTM recurrent networks." The Journal of Machine Learning Research 3 (2003): 115-143.

[19] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[20] Zhang, Xiang, and Yann LeCun. "Text Understanding from Scratch." arXiv preprint arXiv:1502.01710 (2015).

[21] Yu, Kai. "Large-scale deep learning at Baidu." Proceedings of the 22nd ACM international conference on Conference on information & knowledge management. ACM, 2013.

[22] Hinton, Geoffrey E., et al. "Improving neural networks by preventing co-adaptation of feature detectors." arXiv preprint arXiv:1207.0580 (2012).